NYU·poly

POLYTECHNIC INSTITUTE OF NYU

Mechatronics Final Paper

Autonomous Refrigerator Robot

Vinícius Bazan

Adam Jerozolim

Luiz Jollembeck

Table of Contents

Introduction

One of the most common pieces of equipment found in all biomedical and research laboratories are the specimen refrigerator. While seeming not as glamorous or fancy as some of the other laboratory equipment, the specimen refrigerator plays a crucial role in libratory research. Almost all experiment involving living cultures or specimens require them to be maintained at a specific temperature. Sometimes the temperature change can be for preservation purposes and sometimes it might be to simulate an actual environmental condition. Nevertheless, temperature control of experiments is of outmost importance in any modern research facility.

When working with specimens in a laboratory it is often necessary to process samples at specific temperatures and times. In a busy biomedical lab, managing and handling all of the samples can be an overwhelming and daunting task. Different materials with varying densities can have different heat transfer coefficients as well as specific heats. With each sample put into the fridge at a different time there is almost no way to know when each sample will reach its ideal temperature. Some samples might be required to be chilled to near freezing while some might just need their temperatures reduced by a few degrees. A device that could both measure the temperature of the samples, as well as retrieve them from the refrigerator when a desired temperature is reached, would be an ideal sought after device to have in a biomedical laboratory.

The goal of our Mechatronics final project was to design and create a device capable of managing a biomedical laboratory's refrigerated sample inventory. In order to achieve this, the device needed to be able to completely operate on its own without needing the operator's input or commanding. The device would operate autonomously within an enclosed refrigerator and monitor the samples and vials being studied in the lab. When called upon to, the device would be able to retrieve a required sample based either on the time of day, time the specimen was required to be in the refrigerator or even have it removed at a set temperature point.

The flexibility of incorporating a micro controller into the device allows it to be configured to various users' preferences. For instance, a university might be performing a class experiment and require a number of samples to be taken out of the refrigerator and cooled before a class begins. By using our device the teacher can program in a time that the samples are to be ready and the device will automatically make sure they are out and ready for use.

A different researcher might be performing an experiment that requires a culture to be chilled for an exact amount of time. Using our device to make sure the specimen is removed after the proper time will free up the researcher to perform other tasks while waiting for the sample, improving productivity and performance.

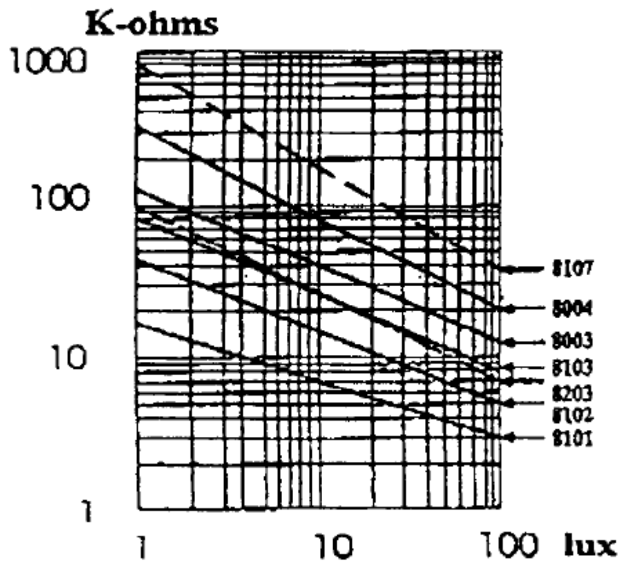
The most complex, and therefore most marketable, use for our device would be in a scenario where a variety of samples and cultures are being studied simultaneously. In a scenario such as this one, where different specimens are required to be controlled at similar temperatures, figuring out exactly how long and at what time each sample will reach its set temperature can be a task that can take a dedicated researcher all day to achieve. By incorporating our device into the lab, the researchers are free to take care of more pressing devices while our machine monitors all of the specimens. Without any effort at all it can not only monitor each of the specimens' temperatures, but remove them from the refrigerator when the desired temperature is reached and place it on the lab table, alerting the researcher that his specimen is ready. This type of control and operation can allow optimal efficiency and productivity in the lab.

Components

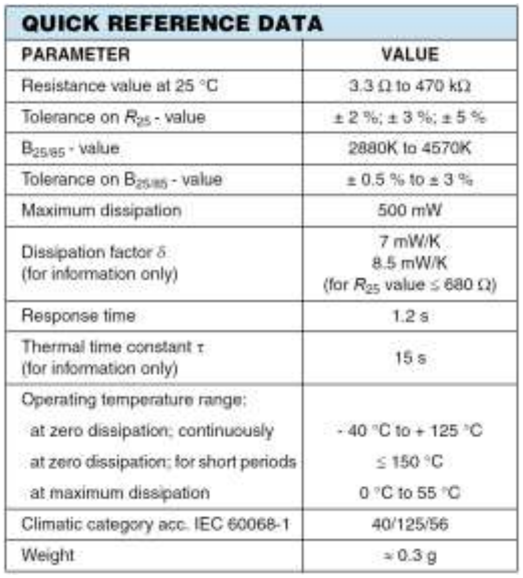
In addition to standard common electronics, our robot incorporated other more complex and wider ranging devices. In this section we will explain how some of those devices work and where they are used.

Photoresistor

In order for the robot to understand when and where a new specimen was placed into the refrigerator, a sensor system needed to be set up. The system allows the robot to identify in what locations and when an object was placed in a given location with the use of a photoresistor. A photoresistor is a resistor that can change its resistance based on a light incident on its surface. A photoresistor placed in a dark environment will have a very large resistance, possibly in the Mega-ohm range, while the same resistor placed in a well lit area can have a resistance as small as one K-ohm. This change causes an almost 1,000 time increase in the resistance of the photoresistor. The photoresistors we selected for our project were Jameco #202403 photoresistors. The operating parameters and characteristics of the photoresistor can been see in the graph to the left.



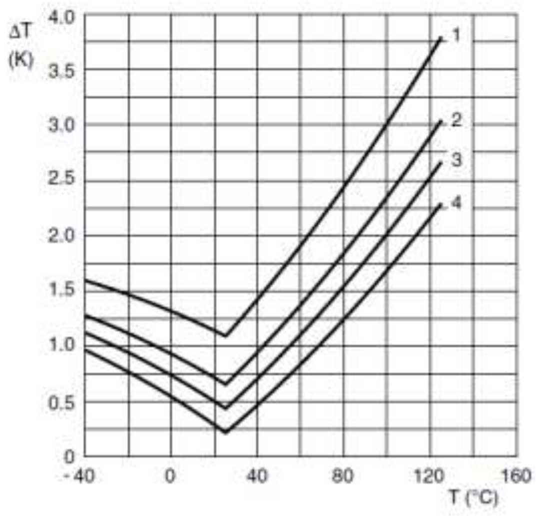
By placing a photoresistor in each of the possible sample locations within the refrigerator, we can get a reading of where and when a sample was placed in the refrigerator. When a sample is placed in the refrigerator it covers one of the photoresistors, thus blocking out the light and increasing its resistance. Since the basic stamp can only operate in a high and low setting, the change in resistance alone is not enough for the basic stamp to recognize when the resistance is raised. Instead, we were able to use an R-C circuit, with the photoresistor acting as the resistor, and the basic stamp function RCTime. When a pin is set high it will discharge a capacitor by providing it with 5v on the V_{ss} side of it. When the pin then goes low, the capacitor begins to charge. When the capacitor voltage reaches 3.6V approximately, the voltage on the Basic Stamp pin is 1.4V and the pin state flips from HIGH to LOW. The time of this process is recorded by the microcontroller. This time is dependent of the resistance of the photoresistor and the capacitance of the capacitor. By measuring the variations in time, we can determine whether the photoresistor is in its high or low resistance state, and thus covered or uncovered, respectively. Figure 5 exemplifies and RC circuit using photoresistor.



QUICK REFERENCE DATA	
PARAMETER	**VALUE**
Resistance value at 25 °C	3.3 Ω to 470 kΩ
Tolerance on R_{25} - value	± 2 %; ± 3 %; ± 5 %
$B_{25/85}$ - value	2880K to 4570K
Tolerance on $B_{25/85}$ - value	± 0.5 % to ± 3 %
Maximum dissipation	500 mW
Dissipation factor δ (for information only)	7 mW/K 8.5 mW/K (for R_{25} value ≤ 680 Ω)
Response time	1.2 s
Thermal time constant τ (for information only)	15 s
Operating temperature range:	
at zero dissipation; continuously	- 40 °C to + 125 °C
at zero dissipation; for short periods	≤ 150 °C
at maximum dissipation	0 °C to 55 °C
Climatic category acc. IEC 60068-1	40/125/56
Weight	≈ 0.3 g

Thermistor

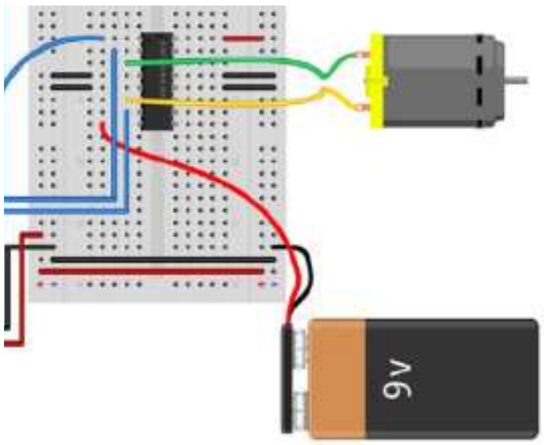
Another component integral to our design involved the use of a thermistor. A thermistor, similarly to the photoresistor, is a variable resistance device. However, what makes the thermistor differ from the photoresistor is that instead of changing resistance relative to the light incident on its surface, the thermistor changes its resistance based on the temperature of its surface. The consequence of that is that when the surface of the thermistor becomes cold, its resistance goes up, and conversely, when the surface of the thermistor becomes hot the resistance goes down. The graph shows the operating characteristics of our thermistor. As the graph reveals, once the thermistor enters its operating range, the relationship between the change in temperature and change in resistance is relatively linear. The thermistor we selected for our project is the SEN-00250.



Similarly to the photoresistor, in order to interface the variable resistor with the basic stamp, an RC circuit has to be constructed and monitored using the RCTime function. By calibrating the thermistor with a know temperature we are able to figure out what temperature the thermistor is sensing based on the return value from the RCTime function.

H-bridge

Another unique component to our project is the use of an h-bridge. When running a DC motor, such as the one we use to operate the robot arm, a large amount of current is needed, especially on start up. If run directly off the basic stamp the current would be too high and would short out the basic stamp's circuitry. To circumvent this from happening a device called an H-bridge is used. The component runs off the 5v source of the basic stamp, but in addition has a separate and isolated power source connected to it to run a device requiring higher currents than the stamp can provide, such as a DC motor. The 5v source from the basic stamp is used to control MOFSET transistors within the H-bridge. By opening and closing specific transistors in specific patterns the H-bridge is capable of directing the higher source current through either direction in the motor. This allows the motor direction to be controlled using only the limited current 5V source from basic stamp. With the Basic Stamp pin control it's possible to set the proper combination of High and Low states at the pins of the H-Bridge in order to make the current flow in one or other direction through the motor, making it run forward or backwards.



Some of the more common components used in our project and not mentioned in this section include LED's to provide a trigger for the photoresistors, resistors and capacitors to complete circuits for the outboard components and servo motors to control the driving motion of the robot and opening and closing of the robotic arm.

Capabilities

The device can be programmed to perform a variety of functions. In this prototype, 3 functions were implemented. They are better explained as follows.

Get the coldest can/sample

In this function, the coldest can/sample among the set is picked up. For each existing sample the temperature is recorded and then compared to the lowest one. After all samples have been tested, the robot gets back to the position of the coldest one, picks it up and puts in the position for the user to get it.

Get a cold can/sample

A cold reference is set. The device searches for the first sample to be in a temperature under this reference temperature. When it is found, the robot picks up the sample and, similarly to the previous function, puts it on the position where the user can get it.

Check freezing

This last function is responsible for checking if the samples are below a pre-set freezing temperature. This is a particular especial function, since the freezing temperature reference can be set to any desired temperature, and thus a researcher is able to have feedback of the actual condition of the samples which temperatures are expected not to go below a certain reference. If any of the samples' temperatures reaches the freezing temperature, a warning is reported to the user.

Bill of materials

The following table contains all the materials used to build the device, as well as their costs.

Component	Quantity	Cost per unit ($)	Cost ($)
Photoresistor	4	1.99	7.96
LED	4	0.50	2
Thermistor	1	1.95	1.95
H-Bridge	1	2.95	2.95
Servo-motors	2	12.99	25.98
DC Motor	1	20	20
Buttons	3	0.50	1.5
Op-Amp IC	1	0.55	0.55
0.01µF Capacitor	3	0.15	0.45
1µ Capacitor	1	0.75	0.75
Resistors	12	0.20	2.4
Battery holder	2	1.99	3.98
AA Batteries	6	2.50	15
Basic Stamp 2	1	49.99	49.99
LCD Display	1	25	25
Board of Education	1	69.99	69.99
Breadboard	2	10	20
Wire pack	1	10	10
K'nex set	1	30	30
Total			290.45

The overall cost was calculated based on the commercial price of each component. If however the device is going to be produced in large scale, the price would probably be much lower. First of all due to large volume discounts for most of the components. Besides that, in a real large scale production, the hardware would have to be modified, since K'nex is very good for prototyping but not for a final product. Thus, 3D print of parts as well as aluminum and steel fabricated sections of the robot should be considered, which would make the hardware itself stronger and even cheaper, since these materials are widely used and their easy handling and fast production contributes cost reduction.

Most of the components used in this project are produced by Parallax. The cost reduction for large quantities is approximately of 10%. The expected cost reduction for the other components can also be expected to follow the same standard. The only possible bigger reduction would be the hardware production since an automated factory as well as cheaper materials yields less costs than K'nex.

In particular, the unit price for a large amount of Basic Stamps is cut by half. For the board of education, the reduction is 20%. Considering all the components and parts involved on the production of the robotic device, the large production price is estimated to be around US$ 220,00.

Circuit

Buttons

Three buttons are present on the system. One of them is used to choose the function that the system will perform, the second is to enter that option and the third one is used as an emergency button. When this last button is pressed, all the functions that the device is performing are stopped. The circuit used for the buttons is shown on Figure 1.

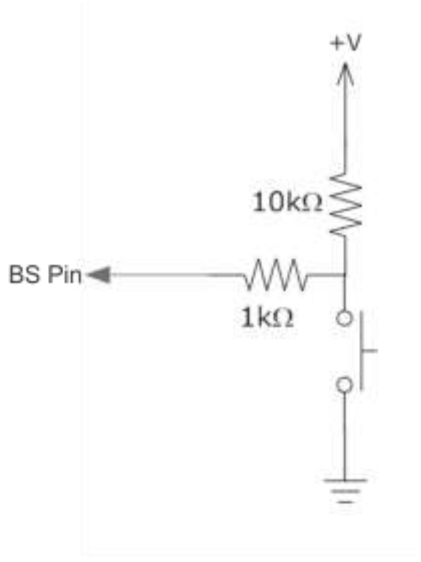


Figure 1: Buttons – NO, Active Low

All three buttons are normally open, active low.

LCD Display

A LCD display is used to show to the user the options that the device can perform and also show warnings, such as "No cans on the fridge". The display used was the one produced by Parallax, 2x16, non-backlit. The connections for this component are shown in Figure 2.

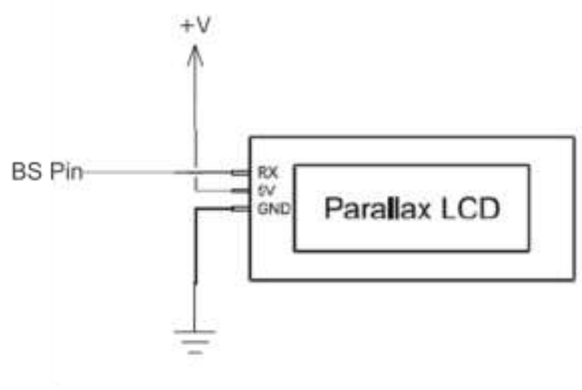


Figure 2: LDC display connections

Servo Motors

Two Parallax servo motors are responsible for two actions: running on the trail and opening/closing the claw to pick up cans or samples. The first one is a continuous servo, while the second has limited angular movement. Both are controlled using PULSOUT command, as better explained on the Programming section. On Figure 3 it is possible to see how they are connected to Basic Stamp.

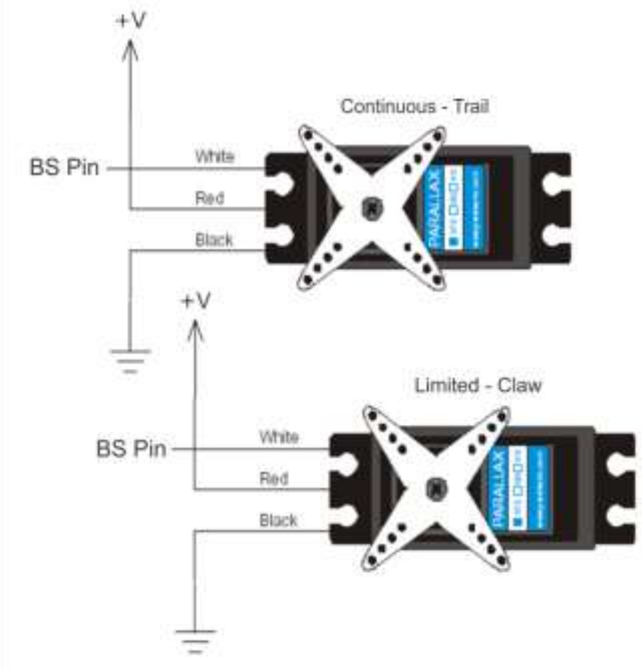


Figure 3: Servo motors connections

DC Motor

For the purpose of putting the device's arm up and down, a DC motor was used. It was chosen, instead of a servo, due to the easier attachment of this motor to the K'nex structure. Furthermore, since for performing opposite actions such as putting the arm up and down requires the motor to be run in opposite directions, an H-Bridge was used to make the current flow either on one or the other direction on the motor. Also, the motor has to be powered by an external source, a 6V battery pack, since it may require more current to work than the Basic Stamp can provide. Also, it was noticed that the voltage of the battery pack had to be greater than the one provided by BS in order to the H-Bridge work properly. Figure 4 presents an schematic of the circuit used for the DC motor.

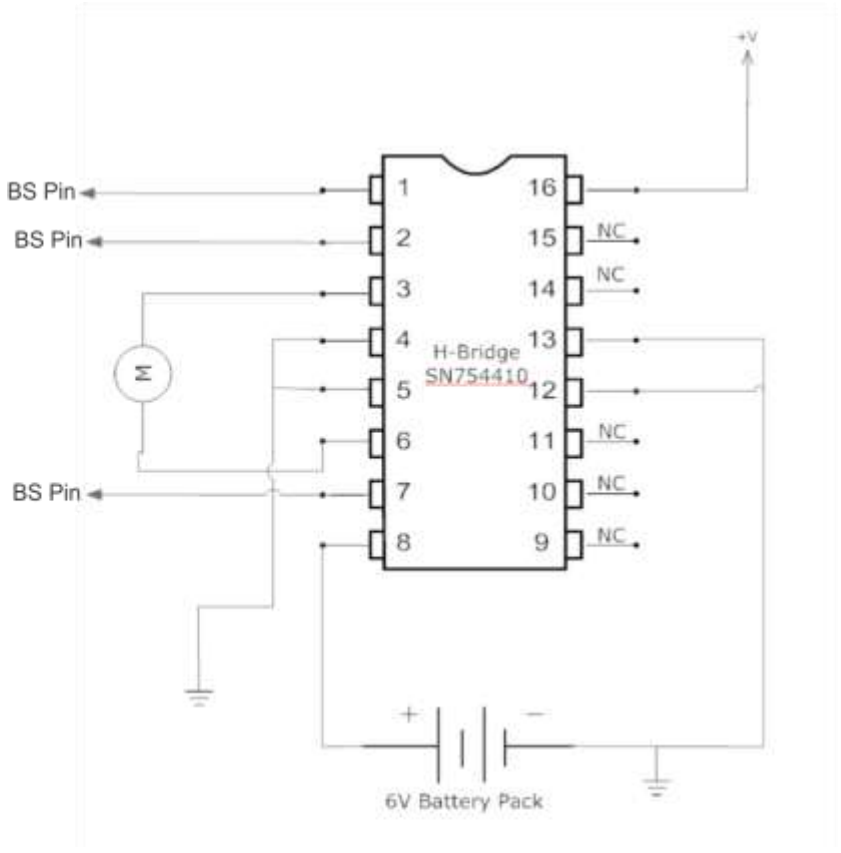


Figure 4: DC Motor + H-Bridge

Position recognizing and sample finding system

Various samples can be placed on the fridge and the device has to know where to stop in order to reach them. Also, a good amount of time can be saved if the device only stops at the positions where there are samples, skipping the ones where there's nothing. Thus, in each position where a sample can be placed there is a photoresistor in a RC circuit. As previously described, when there is no sample in a spot, the resistance of the photoresistor gets lower, due to the presence of light. When a sample is placed, the darkness makes the resistance go higher. With this, it is possible to save the positions where there are samples and avoid stopping on the ones that no sample is placed.

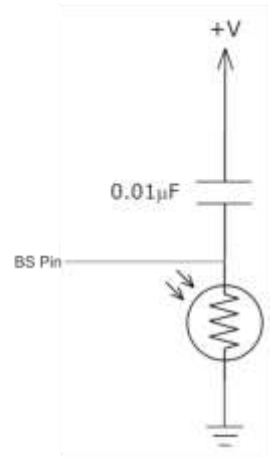


Figure 5: Position photoresistors circuit

Besides that, in front of each position, on the trail, a lit LED is placed so that the device can count how many positions it has passed through. In order to detect a lit LED, another photoresistor, attached to the device, is used. This one, however, is connected to an Op-Amp with positive feedback instead of a RC circuit. The purpose of this is that it would be a bad option to place between every line of code a portion of code to check the time of charging of the capacitor, since it isn't possible to run programs in parallel using BS. Thus, the photoresistor was connected to an Op-Amp circuit working as a comparator. A reference voltage was defined in order to tell if there is LED light or not reaching the photoresistor. Thus, the output of the Op-Amp is either zero or one, when there is LED light or not.

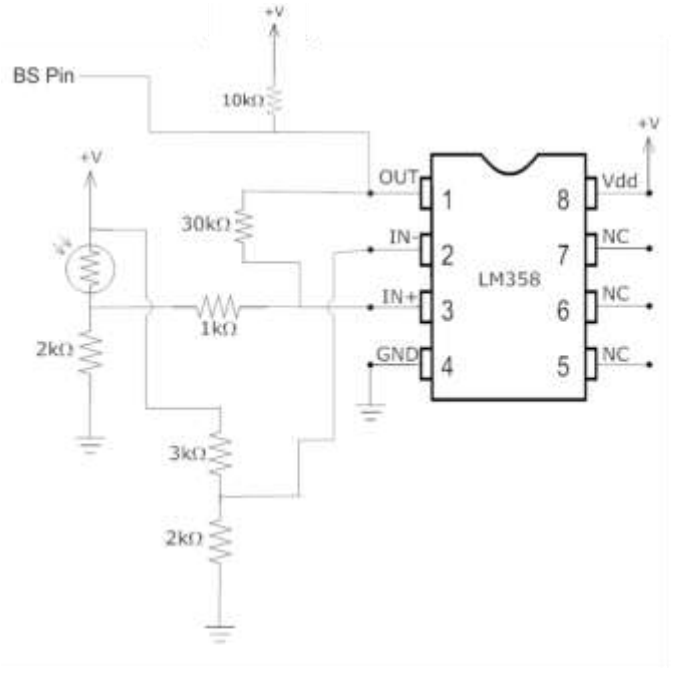


Figure 6: Op-Amp circuit for photoresistor-based LED detecting

This circuit with positive feedback is used to avoid chattering. The resistance of the photoresistor varies approximately from 2.5kΩ to 4kΩ when there is no LED light and when there is, respectively. Thus, this circuit was made in order that the comparison value is 2V and the upper and lower thresholds are, respectively, 1.9V and 2.07V.

The LEDs were connected as shown below. Since in the prototype there are 3 slots, there are 3 LEDs for them and one more to indicate the position where the sample has to be dropped. The LEDs are connected in parallel and are powered by a 3V battery pack.

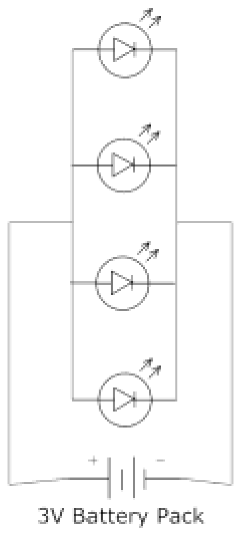


Figure 7: LED connections

Thermistor

The device's temperature sensing is made using a thermistor. Like the position photoresistors, the thermistor is connected in a RC Circuit. It take some seconds to reach steady-state, that is, for the resistance to stop varying. The circuit for the thermistor connection is show in the following figure.

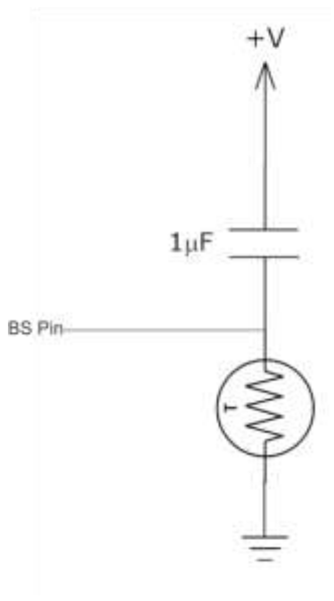


Figure 8: RC circuit – Thermistor

Mathematical background

Resistor-Capacitor circuit – RCTime

As previously mentioned, the RCTime function is capable of recording the time for the capacitor in a RC circuit to charge from 0 to 3.6V. The basic circuit for this purpose is the one of Figure X. Basically, this is a first order system. It can be shown through the use of Laplace and inverse Laplace transform on the circuit's mathematical model that the time for this charging is given by

$$t = -RC * \ln\left(\frac{1.4V}{5V}\right) \qquad (1)$$

Where t is the charging time, R is the resistance value, C, the capacitance, 1.4V is the threshold for High-Low transition on the BS and 5V is the voltage supply on which the RC circuit is connected. Therefore, the time for charging the capacitor varies linearly with the value of R and C, since ln(1.4/5) is a constant.

Op-Amp circuit – Hysteresis circuit

The Op-Amp circuit used in this project is a comparator with positive feedback to avoid chattering. The values of the resistances are chosen in order to set an upper threshold and a lower threshold. Given a comparison reference set as the non-inverting voltage, the upper threshold should be higher than this value and the lower threshold, lower. The proximity between these two values can be configured by just varying the values of the resistances. Given the following circuit,

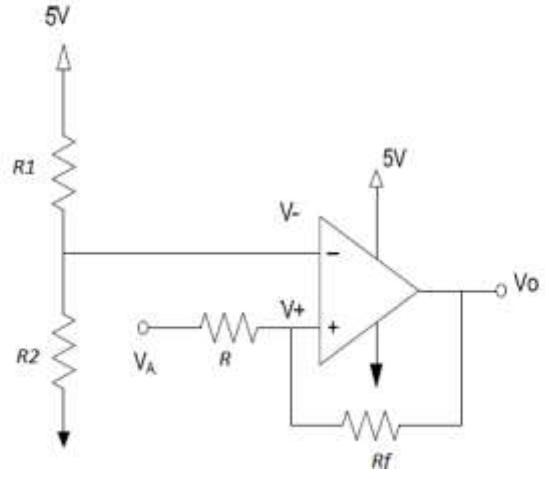


Figure 9: Comparator with positive feedback

it is possible to show that the upper threshold is defined for V_A through the expression

$$\frac{V_oR + V_AR_f}{R + R_f} > V_- \quad (2)$$

Where in this case $V_o = 0V$, since the voltage output should go from low to high. For the lower threshold, the expression would be

$$\frac{V_oR + V_AR_f}{R + R_f} < V_- \quad (3)$$

In this case, $V_o = 5V$, since V_o goes from high to low.

This effect is called hysteresis and is very good to avoid chattering, which could make the entire device's operation go wrong. This effect is better visualized on the next figure.

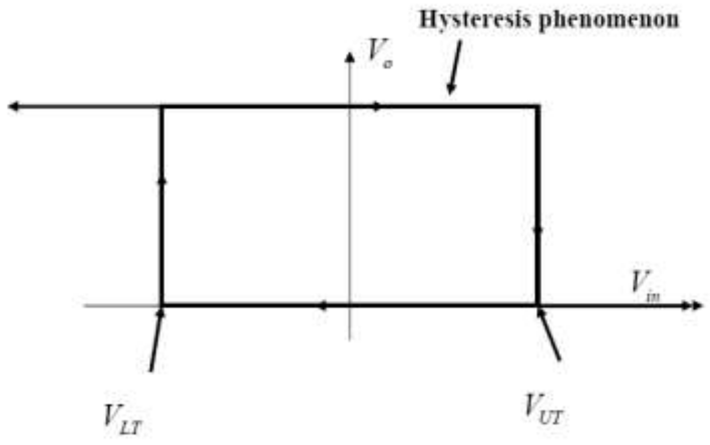


Figure 10: Hysteresis Phenomenon

Current limitation – BS protection

One of the main issues on any project involving the use of a microcontroller like the Basic Stamp which has several general purpose I/O pins is the pin protection. All the connections should be planned not to exceed the maximum current that each pin can handle. When only one pin is used, it is capable of sinking 25mA and sourcing 20mA. When a group of 8 pins is used, the group can sink 50mA and source 40mA. Thus, in order not to exceed the maximum current at a pin, the protection resistor's resistance has to be calculated through

$$R = \frac{V}{I_{max}} \qquad (4)$$

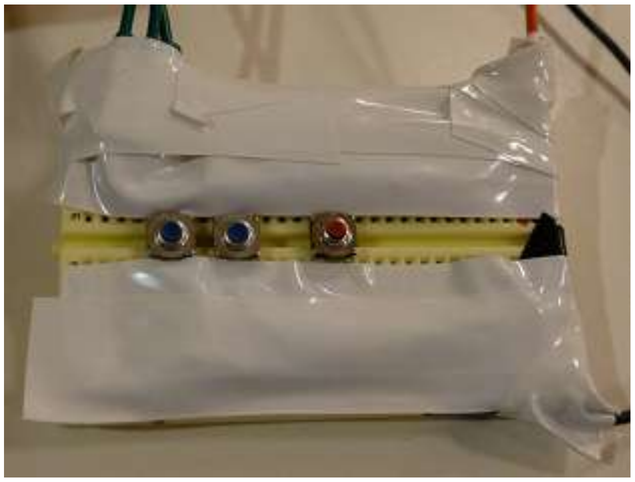
Where V is usually 5V, the BS power supply. And I_{max} is the maximum current that can go into or out a pin, and depends on the components used and the number of pins working at the same time.

Operation

The robotic arm is developed to work in cold, closed environments. For its proper operation, the surroundings must be clean and free of any kind of obstacles that could stop it from moving on the trails, or hitting objects while moving the arm. During the execution of a function, the robot must not be touched or pushed in any ways. Unexpected contact may cause permanent malfunction.

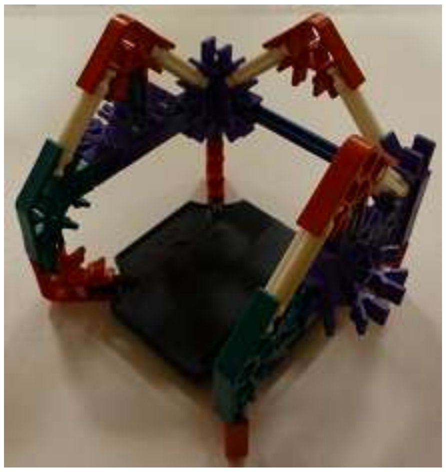
Selecting a Function

To start the operation, the robot must be turned on. This is done by changing the position of the switch from 0 to 1. The LCD display will then list the functions. The user must wait until the message "Select Function" is shown to start pressing the buttons. Button 1 is used to select the function, where pressing the button increases the number until the user reaches the desired number. Button 2 enters the selection and starts operation.



Placing the Cans

To make sure the arm will detect, sense, and lift the cans properly, every can must be placed carefully on the determined spot. There are 3 possible positions, each one with its own circular pad. The user must place the can on the Holder (supplied) and, then, position it adequately on the pad. The arm will automatically detect the positions and perform the selected functions. The Holder makes possible for the robot to lift the can, being an important part of the product.



Emergency Button

Even taking all precautions, errors or misbehaviors may occur. If any malfunction may cause harm to the robot, users or surrounding environment, the emergency button must be pressed. It immediately stops the arm from functioning. To make it keep working from where it stopped, the emergency button must be pressed again. Do not press the reset button!

Programming

The Basic Stamp has a restriction of only 24 bytes of memory for code and variables. To make sure this amount of memory would be enough, the code was built to be as modular as possible. With this, subroutines were created to perform repetitive actions that are called from several parts of the code, reducing its size. The constant branches may slow the process for some microseconds, since the pipeline needs to be flushed every time a branch happens. The arm doesn't have any severe time restriction, though, so the trade-off is reasonable.

A short explanation of some important parts of the code is shown below. The full code can be found under Appendix A.

Main

The main loop of the program is very simple, only calling a sequence of subroutines that are responsible for determining which actions should be taken.

```
 MAIN: 'structure
DO
  GOSUB START          'subroutine that displays the operational options of the robot
'and asks to choose one of them
  GOSUB OPTION         'subroutine in which it is possible to choose the option and
'submit it to the robot controller
  GOSUB CHECKCANS      'subroutine that checks in which positions there are cans and
'in which there are not
  IF POSCANS = 0 THEN GOTO NOCANS      'if no can was found in CHECKCANS, then system
'alarms that there's no can on the fridge
  IF FUNCTION = 1 THEN GOSUB GETCOLDEST   'depending on the option selected, the
'program is redirected to a different subroutine
  IF FUNCTION = 2 THEN GOSUB FIRSTCOLD
  IF FUNCTION = 3 THEN GOSUB FREEZING

LOOP
```

As noticed, it is in loop, which means it will restart after functions is executed.

Start

This is the first subroutine called by Main. It is responsible for showing the user which functions are available, by displaying them and their respective numbers on the LCD screen.

```
START:
PAUSETIME = 10
SEROUT LCDPIN, 84, [22, 12] 'ACTIVATE LCD, CLEAN SCREEN
PAUSE 5
SEROUT LCDPIN, 84, ["Select an", 13, "option:"]
GOSUB PAUSESUB
SEROUT LCDPIN, 84, [12]
PAUSE 5
SEROUT LCDPIN, 84, ["1:COLDEST CAN"]
GOSUB PAUSESUB
SEROUT LCDPIN, 84, [12]
PAUSE 5
SEROUT LCDPIN, 84, ["2:ANY COLD CAN"]
GOSUB PAUSESUB
SEROUT LCDPIN, 84, [12]
PAUSE 5
SEROUT LCDPIN, 84, ["3:CHECK FREEZING"]
GOSUB PAUSESUB
SEROUT LCDPIN, 84, [12]
PAUSE 5
SEROUT LCDPIN, 84, ["Press buttons", 13, "to select"]
GOSUB PAUSESUB
RETURN
```

Option

After the Start subroutine is over, the user is allowed to select the desired function. The Option subroutine checks if any button is pressed, detects it and performs operations accordingly.

```
OPTION:   'SELECTED INSTRUCTION

SEROUT LCDPIN, 84, [12]
PAUSE 5
SEROUT LCDPIN, 84, ["Function: ", DEC FUNCTION]  'prints on the LCD display the number
'of option while it's being selected

DO

  IF ( BUTTON1 <> PRVSVALUE1 ) AND (PRVSVALUE1 = 0)  THEN      'the buttons 1 and 2
'are Normally Open, Active Low
                                                              'button 1 changes
'function and has to be released in order to the command to be processed
    FUNCTION = FUNCTION + 1                                   'this is the purpose of
'checking if the logic state of the button is different of
    IF FUNCTION > 3 THEN FUNCTION = 1                         'the previous one
'(PRVSVALUE)
    SEROUT LCDPIN, 84, [138, DEC FUNCTION]

  ENDIF
  PRVSVALUE1 = BUTTON1
  PAUSETIME = 2
  GOSUB PAUSESUB

LOOP UNTIL BUTTON2 = 0            'When button 2 is pressed, executes the selected
'functionality

SEROUT LCDPIN, 84, [12]
PAUSE 5
SEROUT LCDPIN, 84, ["Function", 13, "selected: ", DEC FUNCTION]   'Keeps displaying
'the function selected while running

RETURN
```

The majority of the subroutine runs inside a LOOP cycle. This checks if button 1 was pressed, incrementing the value of the function to be selected. The PRVSVALUE1 variable is used to determine whether the button was pressed and changed its state to “not pressed”, allowing the program to only detect it once, after the user releases it. The LOOP keeps running until user presses button 2, which will then show the selected function and return to main.

Checkcans

When the function is selected, the arm checks the can spots to recognize which ones have cans and which don’t. This is done by reading the RC circuit and comparing it to a threshold: when the time is higher than the threshold, there is a can, otherwise, no cans are places on that spot.

```
CHECKCANS:   'CHECKS IF THERE ARE CANS ON THE POSITIONS. IF YES, SETS THE BIT OF THE
'VARIABLE TIME REFERRED TO THAT POSITION
```

```
  PAUSETIME = 1

  HIGH POS1                             'RC circuit was used
  GOSUB PAUSESUB
  RCTIME POS1, 1, TIME
  IF TIME > 500 THEN POSCANS = POSCANS | %1

  HIGH POS2
  GOSUB PAUSESUB
  RCTIME POS2, 1, TIME
  IF TIME > 10 THEN POSCANS = POSCANS | %10          'different time values for
'different photoresistors

  HIGH POS3
  GOSUB PAUSESUB
  RCTIME POS3, 1, TIME
  IF TIME > 500 THEN POSCANS = POSCANS | %100

RETURN
```

Due to differences on the photo resistors, each circuit needs a different threshold. The bits of the variable POSCANS are set according to the detected cans. This means that, for example, there's a can on the third spot, the third bit of this variable will be set. This procedure allows the arm to easily check where to stop, as further shown.

Nocans

The Nocans subroutine is very simple. Basically, what is does is check the variable POSCANS. If it is zero, which means no cans were detected, displays an alert message and returns to the beginning of the program.

```
NOCANS:                'If no can is detected, warning is shown

SEROUT LCDPIN, 84, [12]
PAUSE 5
SEROUT LCDPIN, 84, ["NO CANS!"]
PAUSETIME = 10
GOSUB PAUSESUB
GOTO MAIN
```

General Subroutines

These are functions that are used several times through the code. Making them subroutines avoids the need of placing the same repetitive code in different places on the program, saving memory space.

- *Leaveled:*

When the robot stops in a position, it is always sensing that it is above an LED. So, after doing what it should do in that position, it needs to leave the LED area before sensing again when to stop. This function makes the arm move on the trail for a while, making sure that happens.

```
LEAVELED:          'this function makes the movement servo run for a while before
'sensing if it is on a spot or not
FOR X = 0 TO 100   'this makes possible to use DO LOOP UNTIL function to sense
'position
  IF BUTTON3 = 0 THEN GOSUB EMERGENCY
  PULSOUT RUN, FORWARD
  PAUSE 20
NEXT
RETURN
```

- *Armdown:*

Function responsible for making the arm go lower, until the can position. A DC motor is used, so the control is done with an H-Bridge. First we need to enable it, and then select the pins that should be high and low, in order to run the motor to the correct direction. These states should be kept for a determined amount of time. Notice that, when there is a can, the time is lower, since it pushes the arm down.

```
ARMDOWN:         'function that lowers the arm until the can position
HIGH CSARM       'enables the h-bridge
LOW PINARMUP     'sets the right pins so that the motor runs on the desired direction
HIGH PINARMDOWN
IF HASCAN = 1 THEN       'if there is a can, the weight pushes the arm down, needing
'less time to get to the position
  PAUSETIME = 45
  GOSUB PAUSESUB
ELSE
  PAUSETIME = 53
  GOSUB PAUSESUB
ENDIF
LOW CSARM              'disables h-bridge
LOW PINARMDOWN         'sets pin as low

RETURN
```

- *Armup:*

Similar to the Armdown function, but makes the arm go back to the highest position by selecting different pins of the H-Bridge. The time here is higher when there's a can, since it pulls the arm down.

```
ARMUP:           'function that moves the arm up until the highest position
HIGH CSARM       'enables the h-bridge
LOW PINARMDOWN   'sets the right pins so that the motor runs on the desired direction
HIGH PINARMUP
IF HASCAN = 1 THEN      'if there is a can, the weight pulls the arm down, needing more
'time to get to the position
  PAUSETIME = 62
  GOSUB PAUSESUB
ELSE
  PAUSETIME = 61
  GOSUB PAUSESUB
ENDIF
LOW CSARM          'disables h-bridge
LOW PINARMUP       'sets pin as low

RETURN
```

- *Closeclaw/Openclaw:*

To open and close the claw, we basically need to send the servo the desired position. This is used via PWM, determining the time of the high pulse. The servo recognizes and runs until the position set.

```
CLOSECLAW:         'sends PWM pulses to the claw's servo, closing the claw
FOR X = 1 TO 100
  IF BUTTON3 = 0 THEN GOSUB EMERGENCY
  PULSOUT CLAW, 650
  PAUSE 20
NEXT
RETURN
```

The Openclaw function is the same, changing only the value 650 to 300.

- *Pausesub:*

Since an emergency button was implemented, we need to check it all the time to make sure the program stops running whenever it is pressed. Pause commands make the code be stuck for a while, not allowing the button to be sensed. This way, instead of leaving the program frozen on a long pause command, we divided each pause in small 100 milliseconds parts. Every time it loops, it checks if the button was pressed.

```
PAUSESUB:      'function that enables pausing for a defined time while sensing
'emergency button
FOR X = 1 TO PAUSETIME
  PAUSE 100
  IF BUTTON3 = 0 THEN GOSUB EMERGENCY
NEXT
RETURN
```

The variable Pausetime must be set before calling the function. It determines the amount of loops, increasing or decreasing the pause according to the needs.

- *Emergency:*

This is a handler for the act of pushing the emergency button. It doesn't do anything, just waits for the button to be released and pushed again, meaning that the program is ready to go back to where it stopped.

```
EMERGENCY:
DO
LOOP UNTIL BUTTON3 = 1
DO
LOOP UNTIL BUTTON3 = 0
DO
LOOP UNTIL BUTTON3 = 1
RETURN
```

- *Tostart:*

This function takes the arm back to the position zero, where it started. It basically runs the robot backwards until an LED is found. It, then, decrements the position and checks if it's zero. If not, keep running. If yes, checks if there's a can being carried. Drops the can and goes back to main.

```
TOSTART:            'this function leads the arm back to the start position

  FOR X = 0 TO 100   'as in the LEAVELED subroutine, runs for a short period so that
'the robot doesn't get stuck on a LED position
    IF BUTTON3 = 0 THEN GOSUB EMERGENCY
    PULSOUT RUN, BACKWARD
    PAUSE 20
  NEXT

  DO                  'runs until next spot is sensed
    IF BUTTON3 = 0 THEN GOSUB EMERGENCY
    PULSOUT RUN, BACKWARD
    PAUSE 20
  LOOP UNTIL CAN = STOPVALUE

  IF POSITION = 0 THEN      'if position is start point, checks if there is a can
    IF HASCAN = 1 THEN      'if so, drops the can
      GOSUB DROP
      RETURN
    ENDIF
  ELSE                     'else, keeps moving until stop point
    POSITION = POSITION - 1
    GOTO TOSTART
  ENDIF
RETURN
```

The Three Functions

Get the coldest can

This function is divided in two subroutines. The first one, called Getcoldest, is responsible for the movements and checking temperature, until the coldest can is found. The other, Pickcan, is responsible for picking the coldest can after it has been determined. The explanation for some important parts of each subroutine can be seen below.

- *Getcoldest:*

The subroutine will make the arm move on the trail until it finds an LED. It then increments the position variable and compares it to the values of the bits of variable POSCANS, to check if there is a can in that position. If not, loops back to Getcoldest, which makes it go to the next LED spot. If positive, goes on to the next part of the function.

```
   IF POSITION = 1 AND POSITION <> (POSCANS & %1) THEN GOTO GETCOLDEST
   IF POSITION = 2 AND POSITION <> (POSCANS & %10) THEN GOTO GETCOLDEST
   IF POSITION = 3 AND POSITION > (POSCANS & %100) THEN GOTO TOSTART
```

The arm will go down, close the claw, pause for a time for temperature sensing, and check if it's the coldest can. If so, stores the position in variable COLDEST and the temperature time value on the variable LOWERTEMP.

```
  IF LOWERTEMP = 0 THEN        'if this is the first can measured, sets this position
'as the coldest
    LOWERTEMP = TEMP
    COLDEST = POSITION
  ENDIF
  IF TEMP > LOWERTEMP THEN     'else, compares the current temperature value to the
'coldest
    LOWERTEMP = TEMP           'if current is lower, stores position as coldest
    COLDEST = POSITION        'values are in time units, so colder means higher value
  ENDIF
```

The claw will then open, and the arm will go up. If the tested position is the last one in which there's a can, the subroutine Pickcan will be called. Otherwise, goes back to the start of Getcoldest subroutine.

```
IF POSITION = 1 THEN         'checks to know if it should keep moving or pick the can
    IF POSCANS > 1 THEN        'poscans > 1 means that there are cans on other spots,
'so it shouldn't pick
      GOTO GETCOLDEST
    ELSE
      GOTO PICKCAN             'if there are no cans on the other spots, then picks the
can
    ENDIF
  ENDIF
```

Checking the other positions is similar.

- *Pickcan:*

After the position of the coldest can is determined, Pickcan is called to pick it. This is a very simple subroutine. It will run backwards on the trails until the position of the coldest can is reached. Then, the arm will go down, close the claw, go up, and call subroutine Tostart, previously described.

```
IF POSITION = COLDEST THEN      'if the current position is where the coldest can is
'located, picks the can
    GOSUB ARMDOWN
    GOSUB CLOSECLAW
    HASCAN = 1
    GOSUB ARMUP
    GOTO TOSTART                  'sends the robot back to the start position
ENDIF
```

Get the First Cold Can

The procedures taken by this function are similar to the ones on Getcoldest. Since we don't need to store a position, though, there's no need for calling the Pickcan subroutine. The cold can is picked when it is found, branching to the Tostart subroutine to lead the robot back to its initial position and drop the can. If all the cans are sensed and none of them is cold, displays a message warning that there are no cold cans and goes back to starting point.

```
IF POSITION = 1 AND POSCANS = 1 THEN 'checks if there are cans on untested positions
    GOSUB NOCOLDCANS                 'if not, shows message and goes back to start point
    GOTO TOSTART
  ENDIF
```

Procedure for other positions is similar.

Freezing

The movement of this function is just as described previously for the others. This function, though, tests every can available. If the temperature is below the freezing temperature, a warning message will be displayed on the LCD screen, stating that the can is freezing.

```
IF TEMP > FREEZETEMP THEN            'if temperature is lower than reference, warns
'that can is freezing/frozen
    SEROUT LCDPIN, 84, [12]
    PAUSE 5
    SEROUT LCDPIN, 84, ["WARNING!"]
    PAUSETIME = 10
    GOSUB PAUSESUB
    SEROUT LCDPIN, 84, [12]
    PAUSE 5
    SEROUT LCDPIN, 84, ["FREEZING CAN", 13, "ON POSITION ", DEC POSITION]
ENDIF
```

Development

Although prototyped as a “refrigerator insert”, the automatic refrigerator device can be built directly into a larger more spacious laboratory refrigerator. When not being used as an insert, the device can be incorporated into numerous shelves and spaces. This will allow to device to monitor dozens, if not hundreds, of specimens and cultures. One shortcoming of our device was our limited access to microcontroller devices. The basic stamp is build with only a limited number of input and output pins available. This shortcoming severely limits our device’s ability to monitor large amount of samples.

Another manufacturing hurdle that would need to be overcome for a retail prototype would be materials. Our robot was build almost entirely out of K’nex. While using K’nex allowed us to build an operational and fully integrated robot in a short amount of time, it is limited in its compactness and weight. A robot made entirely of aluminum would be able to be built stronger, lighter and smaller. The prototype we designed would require a large refrigerator for only a few samples, while a smaller more compact device would take up a smaller footprint within the refrigerator allowing for more capability. A properly designed and constructed device wouldn’t take up much more space than the samples it is programmed to monitor.

In addition, the temperature of the samples we were demonstrating was taken using a thermistor. While not a rare electrical component, it is not considered to be a mainstream electrical device. This severely limited our abilities to find a device with the response time and range our device fully required. We found our thermistor to have a steady state response time of approximately 20 seconds. While adequate for the prototype, in a retail model a quicker and more accurate thermistor could be used, perhaps one requiring only a few seconds to reach steady state conditions.

Improvements

Even though the emergency button allows the user to stop the execution of the program if a problem occurs, an accidental reset due to random issues would make the robot lose track of its position. The addition of an arm position, a claw state and track position sensors would solve this issue, making possible an initialization routine that would take the robot back to a standard initial state.

Additionally, a better organization of the cables, maybe using a bigger breadboard, would make the prototype a lot better looking, easier to handle, easier to maintain and would avoid problems like wires unintentionally short-circuited.

Conclusion

The refrigerator robot can prove to be an asset to any biomedical or university lab throughout the country. The device can allow researchers to focus on important tasks while taking care of ensuring specimens and cultures are properly taken care of. In addition to increasing productivity and performance, the device can help insure that laboratory errors are left frequent and more controlled. Often a researcher will forget about a sample in a refrigerator or not store it in the precise temperature required. The automatic refrigerator device will remove the various errors associated with precious specimen temperature control. Once properly constructed and manufactured, the automatic specimen refrigerator device is a machine that no laboratory will be found without.

Appendix A – Code

```
' {$STAMP BS2}
' {$PBASIC 2.5}

'Pausetime for functions armup and armdown must be adjusted accordingly

BUTTON1 PIN 0
BUTTON2 PIN 1
CSARM PIN 2
PINARMUP PIN 3
PINARMDOWN PIN 4
THERMPIN PIN 5
BUTTON3 PIN 6
POS1 PIN 7
POS2 PIN 8
POS3 PIN 9
CAN PIN 10

LCDPIN PIN 13
RUN PIN 14
CLAW PIN 15


FORWARD CON 740
BACKWARD CON 760
REFTEMP CON 4000 'Temperature of reference to tell if cold
SENSINGTIME CON 5000
FREEZETEMP CON 10000 'Temperature of reference to tell if freezing
STOPVALUE CON 1

PRVSVALUE1 VAR Bit
FUNCTION VAR Nib
POSITION VAR Nib
COLDEST VAR Nib
TEMP VAR Word
LOWERTEMP VAR Word
HASCAN VAR Bit
X VAR Byte
POSCANS VAR Nib
TIME VAR Word
PAUSETIME VAR Byte

POSCANS = 0
LOWERTEMP = 0
POSITION = 0
FUNCTION = 1
PRVSVALUE1 = 1
COLDEST = 0

MAIN: 'structure
DO
  LOWERTEMP = 0
  COLDEST = 0
  GOSUB START                              'subroutine that displays the operational
'options of the robot and aks to choose one of them
  GOSUB OPTION                             'subroutine in which it is possible to
'choose the option and submit it to the robot controller
  GOSUB CHECKCANS                          'subroutine that checks in which positions
'there are cans and in which there are not
  IF POSCANS = 0 THEN GOTO NOCANS          'if no can was found in CHECKCANS, then
'system alarms that there's no can on the fridge
```

```
  IF FUNCTION = 1 THEN GOSUB GETCOLDEST     'depending on the option selected, the
'program is redirected to a different subroutine
  IF FUNCTION = 2 THEN GOSUB FIRSTCOLD
  IF FUNCTION = 3 THEN GOSUB FREEZING
LOOP


START:
PAUSETIME = 10
SEROUT LCDPIN, 84, [22, 12] 'ACTIVATE LCD, CLEAN SCREEN
PAUSE 5
SEROUT LCDPIN, 84, ["Select an", 13, "option:"]
GOSUB PAUSESUB
SEROUT LCDPIN, 84, [12]
PAUSE 5
SEROUT LCDPIN, 84, ["1:COLDEST CAN"]
GOSUB PAUSESUB
SEROUT LCDPIN, 84, [12]
PAUSE 5
SEROUT LCDPIN, 84, ["2:ANY COLD CAN"]
GOSUB PAUSESUB
SEROUT LCDPIN, 84, [12]
PAUSE 5
SEROUT LCDPIN, 84, ["3:CHECK FREEZING"]
GOSUB PAUSESUB
SEROUT LCDPIN, 84, [12]
PAUSE 5
SEROUT LCDPIN, 84, ["Press buttons", 13, "to select"]
GOSUB PAUSESUB
RETURN


OPTION:   'SELECTED INSTRUCTION

SEROUT LCDPIN, 84, [12]
PAUSE 5
SEROUT LCDPIN, 84, ["Function: ", DEC FUNCTION]               'prints on the LCD
'display the number of option while it's being selected

DO

  IF ( BUTTON1 <> PRVSVALUE1 ) AND (PRVSVALUE1 = 0)  THEN      'the buttons 1 and 2
'are Normally Open, Active Low
                                                               'button 1 changes
'function and has to be released in order to the command to be processed
    FUNCTION = FUNCTION + 1                                    'this is the purpose of
'checking if the logic state of the button is different of
    IF FUNCTION > 3 THEN FUNCTION = 1                          'the previous one
(PRVSVALUE)
    SEROUT LCDPIN, 84, [138, DEC FUNCTION]

  ENDIF
  PRVSVALUE1 = BUTTON1
  PAUSETIME = 2
  GOSUB PAUSESUB

LOOP UNTIL BUTTON2 = 0                                          'When button 2 is
'pressed, executes the selected functionality

SEROUT LCDPIN, 84, [12]
PAUSE 5
SEROUT LCDPIN, 84, ["Function", 13, "selected: ", DEC FUNCTION]   'Keeps displaying
'the function selected while running
```

```
RETURN

CHECKCANS:                                    'checks if there are cans on the
'positions. If yes, sets the bit of the variable time referred to that position
  PAUSETIME = 1
  POSCANS = 0

  HIGH POS1                                   'RC circuit was used
  GOSUB PAUSESUB
  RCTIME POS1, 1, TIME
  IF TIME > 500 THEN POSCANS = POSCANS | %1

  HIGH POS2
  GOSUB PAUSESUB
  RCTIME POS2, 1, TIME
  IF TIME > 10 THEN POSCANS = POSCANS | %10     'different time values for different
'photoresistors

  HIGH POS3
  GOSUB PAUSESUB
  RCTIME POS3, 1, TIME
  IF TIME > 300 THEN POSCANS = POSCANS | %100

  DEBUG DEC POSCANS,CR

RETURN

NOCANS:                'If no can is detected, warning is shown

SEROUT LCDPIN, 84, [12]
PAUSE 5
SEROUT LCDPIN, 84, ["NO CANS!"]
PAUSETIME = 10
GOSUB PAUSESUB
GOTO MAIN

GETCOLDEST:           'in this functionality, the robot checks the temperature of
'every can and gets the coldest one

  GOSUB LEAVELED
  DO                        'runs until detects that it is in a can spot
    DEBUG BIN CAN, CR
    PULSOUT RUN, FORWARD
    PAUSE 20
    IF BUTTON3 = 0 THEN GOSUB EMERGENCY
  LOOP UNTIL CAN = STOPVALUE

  POSITION = POSITION + 1

  IF POSITION = 1 AND POSITION <> (POSCANS & %1) THEN GOTO GETCOLDEST   'Checks if
'there is a can in this position
  IF POSITION = 2 AND POSITION <> (POSCANS & %10) THEN GOTO GETCOLDEST
  IF POSITION = 3 AND POSITION > (POSCANS & %100) THEN GOTO TOSTART    'shouldn't
'reach this point. Just for safety

  GOSUB ARMDOWN

  GOSUB CLOSECLAW

  PAUSETIME = SENSINGTIME/100       'waits for steady state sensing
  GOSUB PAUSESUB
  HIGH THERMPIN                 'temperature sensing
  PAUSE 100
  RCTIME THERMPIN, 1, TEMP

  IF LOWERTEMP = 0 THEN       'if this is the first can measured, sets this position
'as the coldest
    LOWERTEMP = TEMP
    COLDEST = POSITION
  ENDIF
  IF TEMP > LOWERTEMP THEN    'else, compares the current temperature value to the
'coldest
    LOWERTEMP = TEMP          'if current is lower, stores position as coldest
    COLDEST = POSITION        'values are in time units, so colder means higher value
  ENDIF

  GOSUB OPENCLAW

  GOSUB ARMUP

  IF POSITION = 1 THEN        'checks to know if it should keep moving or pick the can
    IF POSCANS > 1 THEN       'poscans > 1 means that there are cans on other spots,
'so it shouldn't pick
      GOTO GETCOLDEST
    ELSE
      GOTO PICKCAN            'if there are no cans on the other spots, then picks the
'can
    ENDIF
  ENDIF
  IF POSITION = 2 THEN        'same logic as previously described
    IF POSCANS > 3 THEN
      GOTO GETCOLDEST
    ELSE
      GOTO PICKCAN
    ENDIF
  ENDIF
  IF POSITION = 3 THEN
      HASCAN = 0
      GOTO PICKCAN
  ENDIF

  GOTO GETCOLDEST

LEAVELED:          'this function makes the movement servo run for a while before
'sensing if it is on a spot or not
FOR X = 0 TO 100   'this makes possible to use DO LOOP UNTIL function to sense
'position
  IF BUTTON3 = 0 THEN GOSUB EMERGENCY
  PULSOUT RUN, FORWARD
  PAUSE 20
NEXT
RETURN

PICKCAN:          'function that does the logic for picking the can when getcoldest is
'selected

  IF POSITION = COLDEST THEN      'if the current position is where the coldest can is
'located, picks the can
    GOSUB ARMDOWN
    GOSUB CLOSECLAW
    HASCAN = 1
    GOSUB ARMUP
    GOTO TOSTART                  'sends the robot back to the start position
  ENDIF
  FOR X = 0 TO 100        'as in the LEAVELED subroutine, runs for a short period so
'that the robot doesn't get stuck on a LED position
    IF BUTTON3 = 0 THEN GOSUB EMERGENCY
    PULSOUT RUN, BACKWARD
    PAUSE 20
  NEXT

  DO                      'keeps moving until next position
    IF BUTTON3 = 0 THEN GOSUB EMERGENCY
    PULSOUT RUN, BACKWARD
    PAUSE 20
  LOOP UNTIL CAN = STOPVALUE

  POSITION = POSITION - 1

  GOTO PICKCAN

DROP:          'drops the can
  GOSUB ARMDOWN
  GOSUB OPENCLAW
  GOSUB ARMUP
  HASCAN = 0

  RETURN


FIRSTCOLD:     'in this subroutine, the robot gets the first cold can it finds,
'comparing the can's temperature to a referece one

  GOSUB LEAVELED
  DO                'moves until next position
    IF BUTTON3 = 0 THEN GOSUB EMERGENCY
    PULSOUT RUN, FORWARD
    PAUSE 20
  LOOP UNTIL CAN = STOPVALUE

  POSITION = POSITION + 1

  IF POSITION = 1 AND POSITION <> (POSCANS & %1) THEN GOTO FIRSTCOLD   'checks if
'there is a can in this position
  IF POSITION = 2 AND POSITION <> (POSCANS & %10) THEN GOTO FIRSTCOLD
  IF POSITION = 3 AND POSITION > (POSCANS & %100) THEN GOTO TOSTART

  GOSUB ARMDOWN

  GOSUB CLOSECLAW

  PAUSETIME = SENSINGTIME/100    'time for the Thermistor to reach steady state
'measure
  GOSUB PAUSESUB

  HIGH THERMPIN                 'temperature sensing
  PAUSE 100
  RCTIME THERMPIN, 1, TEMP

  DEBUG DEC TEMP, CR

  IF TEMP > REFTEMP THEN            'if can is cold enough, picks it
    HASCAN = 1
    GOSUB ARMUP
    GOTO TOSTART
  ENDIF
  GOSUB OPENCLAW
  GOSUB ARMUP

  IF POSITION = 1 AND POSCANS = 1 THEN    'checks if there are cans on untested
'positions
    GOSUB NOCOLDCANS                      'if not, shows message and goes back to start
'point
    GOTO TOSTART
  ENDIF

  IF POSITION = 2 AND POSCANS <= 3 THEN
    GOSUB NOCOLDCANS
    GOTO TOSTART
  ENDIF

  IF POSITION = 3 THEN
    GOSUB NOCOLDCANS
    GOTO TOSTART
  ENDIF

  GOTO FIRSTCOLD

RETURN

NOCOLDCANS:          'function that cleans the lcd and shows message saying there are
'no cold cans

SEROUT LCDPIN, 84, [12]
PAUSE 5
SEROUT LCDPIN, 84, ["NO COLD CANS!"]

RETURN

TOSTART:            'this function leads the arm back to the start position

  FOR X = 0 TO 100   'as in the LEAVELED subroutine, runs for a short period so that
'the robot doesn't get stuck on a LED position
    IF BUTTON3 = 0 THEN GOSUB EMERGENCY
    PULSOUT RUN, BACKWARD
    PAUSE 20
  NEXT

  DO                 'runs until next spot is sensed
    IF BUTTON3 = 0 THEN GOSUB EMERGENCY
    PULSOUT RUN, BACKWARD
    PAUSE 20

    DEBUG BIN CAN, CR

  LOOP UNTIL CAN = STOPVALUE

  IF POSITION = 0 THEN     'if position is start point, checks if there is a can
    IF HASCAN = 1 THEN     'if so, drops the can
      GOSUB DROP
      RETURN
    ENDIF
  ELSE                    'else, keeps moving until stop point
    POSITION = POSITION - 1
    GOTO TOSTART
  ENDIF
RETURN
```

```
FREEZING:       'in this subroutine, the robot keeps checking if one or more cans is
'likely to freeze, that is, below a freezing reference temperature

  GOSUB LEAVELED
  DO
    IF BUTTON3 = 0 THEN GOSUB EMERGENCY
    PULSOUT RUN, FORWARD
    PAUSE 20
  LOOP UNTIL CAN = STOPVALUE

  POSITION = POSITION + 1

  IF POSITION = 1 AND POSITION <> (POSCANS & %1) THEN GOTO FREEZING   'checks if there
'is a can in this position
  IF POSITION = 2 AND POSITION <> (POSCANS & %10) THEN GOTO FREEZING
  IF POSITION = 3 AND POSITION > (POSCANS & %100) THEN GOTO TOSTART

  GOSUB ARMDOWN

  GOSUB CLOSECLAW

  PAUSETIME = SENSINGTIME/100      'time for the Thermistor reach steady state measure
  GOSUB PAUSESUB

  HIGH THERMPIN                 'temperature sensing
  PAUSE 100
  RCTIME THERMPIN, 1, TEMP

  GOSUB OPENCLAW

  GOSUB ARMUP

  IF TEMP > FREEZETEMP THEN                 'if temperature is lower than reference,
'warns that can is freezing/frozen
    SEROUT LCDPIN, 84, [12]
    PAUSE 5
    SEROUT LCDPIN, 84, ["WARNING!"]
    PAUSETIME = 10
    GOSUB PAUSESUB
    SEROUT LCDPIN, 84, [12]
    PAUSE 5
    SEROUT LCDPIN, 84, ["FREEZING CAN", 13, "ON POSITION ", DEC POSITION]
  ENDIF

  IF (POSITION = 1 AND POSCANS = 1) THEN      'checks if there are any untested cans
    GOTO TOSTART                            'if not, goes to start
  ENDIF

  IF (POSITION = 2 AND POSCANS <= 3) THEN
    GOTO TOSTART
  ENDIF

  IF (POSITION = 3) THEN
    GOTO TOSTART
  ENDIF

  GOTO FREEZING                        'else, keeps testing


RETURN

ARMDOWN:               'function that lowers the arm until the can position
```

```
HIGH CSARM              'enables the h-bridge
LOW PINARMUP            'sets the right pins so that the motor runs on the desired
'direction
HIGH PINARMDOWN
IF HASCAN = 1 THEN      'if there is a can, the weight pushes the arm down, needing
'less time to get to the position
  PAUSETIME = 45
  GOSUB PAUSESUB
ELSE
  PAUSETIME = 53
  GOSUB PAUSESUB
ENDIF
LOW CSARM               'disables h-bridge
LOW PINARMDOWN          'sets pin as low
RETURN

ARMUP:          'function that moves the arm up until the highest position
HIGH CSARM      'enables the h-bridge
LOW PINARMDOWN  'sets the right pins so that the motor runs on the desired direction
HIGH PINARMUP
IF HASCAN = 1 THEN      'if there is a can, the weight pulls the arm down, needing more
'time to get to the position
  PAUSETIME = 62
  GOSUB PAUSESUB
ELSE
  PAUSETIME = 61
  GOSUB PAUSESUB
ENDIF
LOW CSARM         'disables h-bridge
LOW PINARMUP      'sets pin as low
RETURN

CLOSECLAW:                              'sends PWM pulses to the claw's servo,
'closing the claw
FOR X = 1 TO 100
  IF BUTTON3 = 0 THEN GOSUB EMERGENCY
  PULSOUT CLAW, 650
  PAUSE 20
NEXT
RETURN

OPENCLAW:          'sends PWM pulses to the claw's servo, closing the claw
FOR X = 1 TO 100
  IF BUTTON3 = 0 THEN GOSUB EMERGENCY
  PULSOUT CLAW, 300
  PAUSE 20
NEXT
RETURN

PAUSESUB:                               'function that enables pausing for a defined
'time while sensing emergency button
FOR X = 1 TO PAUSETIME
  PAUSE 100
  IF BUTTON3 = 0 THEN GOSUB EMERGENCY
NEXT
RETURN

EMERGENCY:
DO
LOOP UNTIL BUTTON3 = 1
DO
LOOP UNTIL BUTTON3 = 0
DO
```

```
LOOP UNTIL BUTTON3 = 1
RETURN
```